Custom Display Lists

Objectives of Module

In this module you will learn how to use special built-in features of the Atari Computer that allow you to mix different graphics modes on one screen. You will explore how to create your own display lists, how to organize screen RAM, and how to use this knowledge to make screen images exactly the way you want them.

Overview (subtopics)

1. Purpose of a Display List.
   Relationship between display lists and graphics modes.
   Display list instructions.

2. What is Screen RAM?
   Relationship between display lists, screen RAM, and graphics modes.
   How to put information into screen RAM.

3. Customized Display Lists.
   What are scan lines and mode lines?
   Combining different mode lines.
   Organizing and using screen RAM.

4. Summary and Challenges.

Prerequisite Understanding Necessary

1. You must be familiar with how to individually use different BASIC graphics modes to put text or graphics on your screen.

2. You must know the purpose of the PEEK and POKE statements and how to use them in BASIC.

Materials Needed

1. BASIC Cartridge.

2. Advanced Topics Diskette.

What is a Display List?

In this section you will learn what a display list is and how to use a display list to specify different graphics modes on the screen.

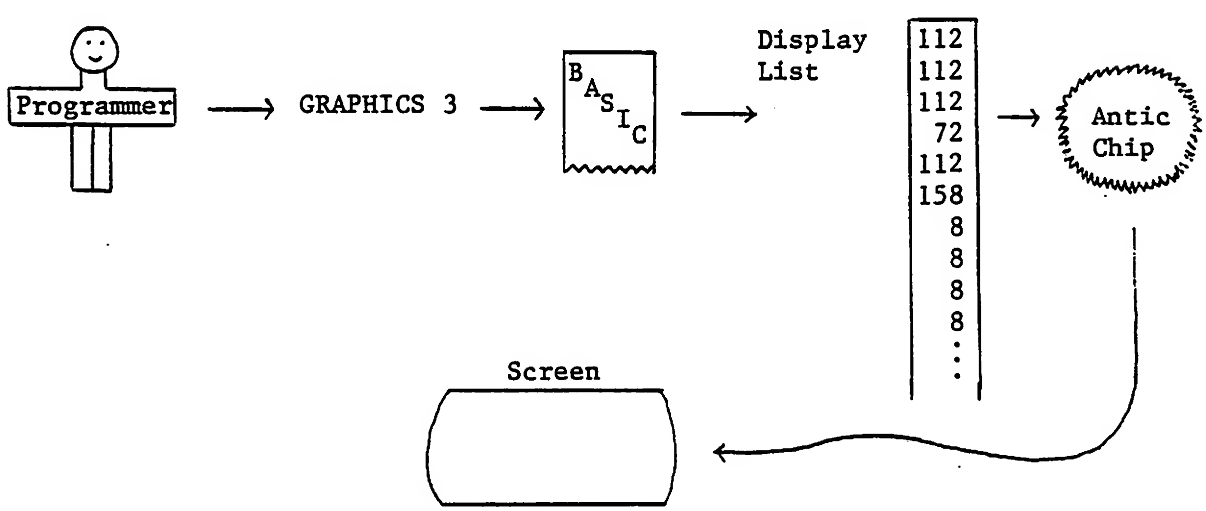
The Atari Computer is extremely versatile. It can print information on the screen in many different ways. When you first sit down at the computer, you see the screen displays text in small letters -- this is graphics mode 0. You can, however, choose to have text printed in larger characters (graphics modes 1 and 2), or you can draw pictures on the screen using other graphics modes.

If you are using the BASIC language, you can tell the computer how to display information on the screen by using the GRAPHICS #n command. Thus, if you say "GRAPHICS 3", the computer knows that it should only show lighted spots (pixels) of up to 4 different colors (if you count the background color) on the screen. The computer must do a great deal of work between the instant you type "GRAPHICS 3" and when the screen is ready to be used for plotting and drawing colors. Luckily, BASIC does all of this hard work for you. In fact, one might say that BASIC simply hands to the Antic chip an entire display list (whatever that is) ready for use. Without this display list, the computer wouldn't function.

In Diagram #1 you can see that a display list is simply a list of numbers. Soon we'll discuss what these numbers mean. First, let's see what kinds of display lists BASIC creates for the various graphics modes. To obtain this list using BASIC, you can tell the computer to go into a particular graphics mode, then save the numbers from a place in memory where the display list is stored, and, finally, return to Graphics 0 and print these numbers out on the screen. The program below accomplishes this task. Type RUN "D:DLIST". Then use this program to answer the questions on Display List Worksheet #1.

```
100 REM *      Program DLIST
110 REM *
120 DIM DLIST(250):DIM SPLIT$(1)
130 GRAPHICS 0
140 PRINT "I will show you the Display List"
150 PRINT "of any graphics mode."
160 PRINT:PRINT "Which mode (0 - 11)";:INPUT MODE
170 IF MODE=0 OR MODE>8 THEN 200
180 PRINT "Split screen (Y or N)";:INPUT SPLIT$
190 IF SPLIT$<>"Y" THEN MODE=MODE+16
200 GRAPHICS MODE
210 REM *
220 REM *      Calculate the memory location
230 REM *      of the display list.
240 REM *
250 LBYTE=PEEK(560):HBYTE=PEEK(561):LOC=LBYTE+256*HBYTE
260 COUNT=0
270 DLIST(COUNT)=PEEK(LOC+COUNT)
280 REM *
290 REM *      The next statement determines if we are
300 REM *      at the end of the display list.
310 REM *
320 IF PEEK(LOC-COUNT-2)<>65 THEN COUNT=COUNT+1:GOTO 270
330 GRAPHICS 0:SPACE=INT(40/(1+INT(COUNT/21)))
340 FOR I=0 TO COUNT
350 POSITION SPACE*INT(I/21)+1,I-21*INT(I/21)
360 PRINT DLIST(I);
370 NEXT I
380 POSITION 2,21:GOTO 160
```

Diagram 1



Display List Worksheet #1

When you use a split screen graphics mode, the top portion of the screen uses one graphics mode while the bottom four rows use Graphics 0. In fact, it is possible to divide a screen into many more pieces, or rows, and make each row, or combination of rows, a different graphics mode.

The numbers in a display list each have a meaning, but you will notice that most of the numbers in any display list are all the same. The number which appears most often tells the computer which graphics mode each horizontal line on the screen represents. You will also notice that this most frequently used number is a different number from the one used to tell BASIC what graphics mode you want. The numbers used in display lists are called Antic mode numbers.

Use the DLIST program to find the Antic modes for each BASIC mode below. Be sure to ask for full screen and not split screen modes. For BASIC modes 0 through 3, count how many times the Antic mode number appears in the display list. (Ask for help if you are confused).

BASIC Mode	0	1	2	3	4	5	6	7	8
Antic Mode									
Number of times									
Number of lines of text or graphics on a full screen	24	24	12	24	48	48	96	96	192

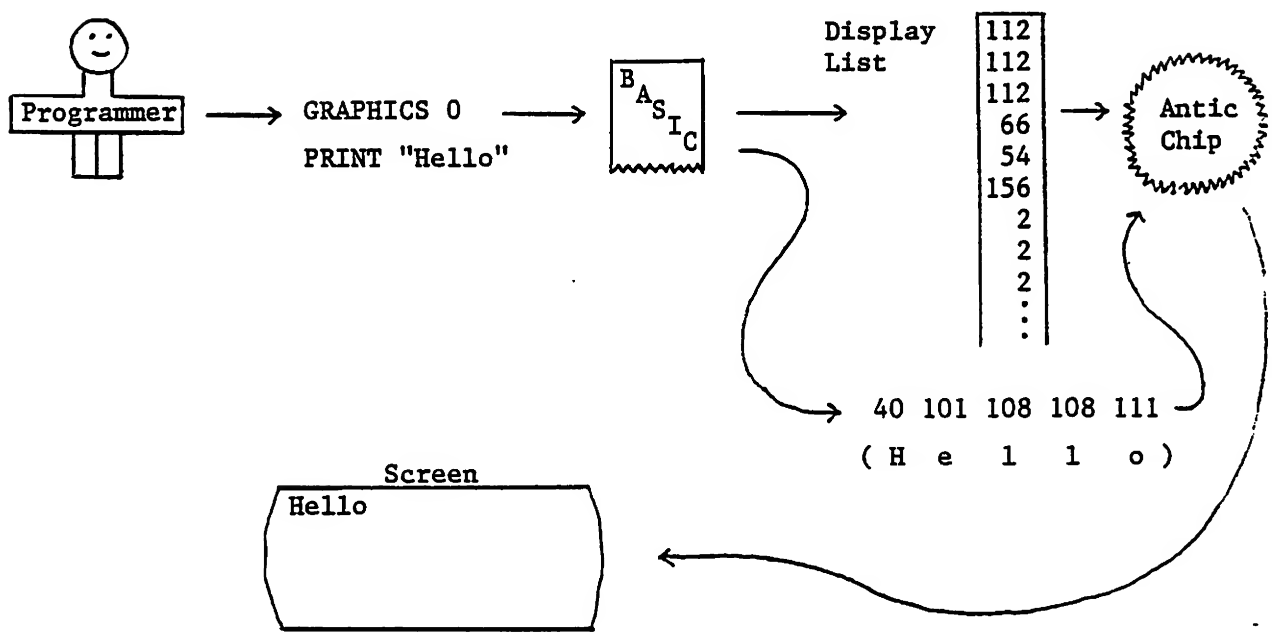
Now look at the display lists for the split screen modes. Which Antic mode appears at the end of these lists (it appears 4th, 5th, and 6th from the end)? _____
Which BASIC mode does this correspond to? _____
How many times does it appear? _____
Recall that we always get 4 text lines in our split screen window.

You are probably wondering why we always seem to be short by one when we count the number of times each Antic mode number appears. The missing mode number is really there, but it is hidden from us. Look at the display list for graphics mode 3, split screen. The fourth number in the list is 72. If we subtract 64 (64 is a special number we'll explore later), we get 72-64 = 8. Aha! -- An extra 8. Now look at the end of the list of 8's just before we switch to 2. You see the number 66. Again, if we subtract 64, we get 66-64 = 2. There's our missing 2. Look again at some other graphics modes and be sure you can find the missing Antic mode numbers.

What is Screen RAM?

In this section you will learn how the computer knows where to store and retrieve information it puts on your screen. You will explore how to change what you see on the screen by directly changing what is stored in memory.

Diagram 2



The display list not only tells the Antic chip what kind of information to put on the screen (small text, large text, graphics), but also tells Antic where to find the information that is going to go on the screen. When you tell BASIC to PRINT "Hello", BASIC puts the word "Hello" in a particular place in computer memory, and Antic knows just where to find it so that it can copy it onto the screen (see Diagram 2). Antic knows where to find it because the display list tells it where to look. The portion of memory where Antic looks to get information to display on the screen is called <u>screen RAM</u> (<u>R</u>andom <u>A</u>ccess <u>M</u>emory).

Remember that special number 64 which helped to hide one of our Antic mode numbers? Look at the Graphics 0 display list in Diagram 2. The fourth number is 66 which is 64+2. That's our one missing Antic mode 2 (BASIC mode 0) number. The 64 has the special meaning of telling the Antic chip that the next two numbers tell where the screen RAM starts. Those numbers are 64 and 156. To get the starting location of screen RAM we always multiply the second number by 256 and add the first. Thus, screen RAM starts at 256*156 + 64 = 40000. Thus, memory locations beginning at 40000 (and, in the case of Graphics 0, ending at 40959) are used to save information that will appear on the screen. (Warning: 40000 is the beginning of screen RAM for Graphics 0 only. Different numbers appear in the different display lists for other graphics modes).

Take the time now to do Display List Worksheet #2.

Display List Worksheet #2

By playing a little with PEEKs and POKEs, you can do some interesting things to screen RAM to make interesting things happen on the screen. Start by following these steps:

1. Press SHIFT-CLEAR to clear the screen. The cursor should be positioned at the top of the screen. It will be automatically positioned two spaces in from the left-most edge.

2. On this top line type your name. <u>Don't press RETURN</u>. Use the arrow keys to move the cursor down a couple of lines and to the left margin (2 spaces in from the screen edge).

3. Be sure you are in the capital letters mode by pressing SHIFT-CAPS. Now type: LOC=40000 and press RETURN. LOC contains the starting address of screen RAM.

4. The first two locations in screen RAM are blank because your name is indented by two spaces. Thus if you type: PRINT PEEK(LOC),PEEK(LOC+1) you will see two zeros.

5. Try typing the following (always press RETURN after each line).

```
PRINT PEEK(LOC+2),PEEK(LOC+3)
POKE LOC+20,PEEK(LOC+2)
```

(Notice what happens on the top line).

```
POKE LOC+40,PEEK(LOC+3)
```

6. Play with the ideas above for a while to explore the possibilities. For example, try things like: POKE LOC+40,165. Try other numbers.

You can also explore screen RAM in other graphics modes, but first you need to find out where screen RAM is located after you enter the graphics mode. Type the following exactly as shown below (no line numbers):

```
GRAPHICS 3
DLIST=256*PEEK(561)+PEEK(560)
```

(This is the location in memory of the display list).

```
LOC=256*PEEK(DLIST+5)+PEEK(DLIST+4)
```

(Here we use the two numbers in the display list which tell us where screen RAM begins).

Now try some of the following and then experiment on your own:

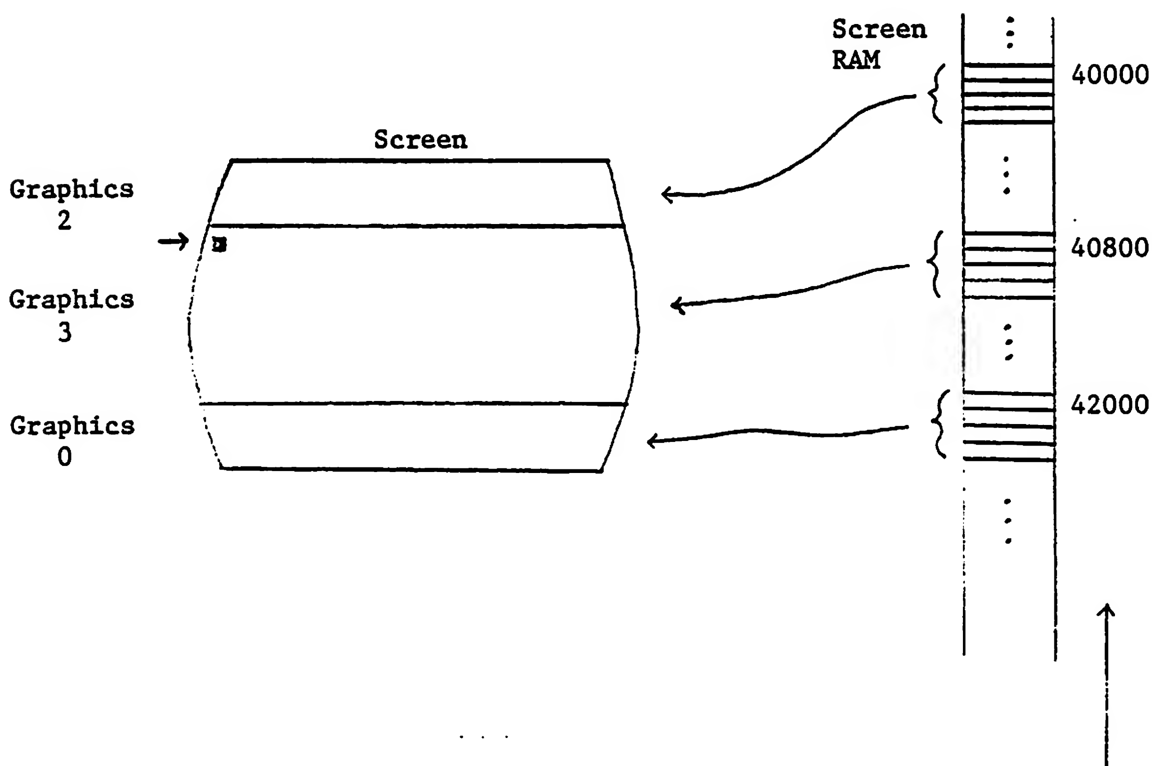
```
POKE LOC,255
POKE LOC,39
FOR I=0 TO 255:POKE LOC,I:FOR J=1 TO 30:NEXT J:NEXT I
FOR I=0 TO 255:POKE LOC+I,I:FOR J=1 TO 20:NEXT J:NEXT I
```

If you are interested in understanding how the computer uses the number in screen RAM to decide what colors to put on the screen, you should study the advanced module on "Internal Representation of Text and Graphics".

You can also put information onto the screen by letting BASIC do it for you, which is, of course, much easier. When you use the standard graphics modes you don't need to do anything special; you just use standard BASIC statements like PLOT, DRAWTO, PRINT #6, etc.

However, when you construct your own display lists, you have to do some special things before you can use BASIC statements to draw or print things on the screen. The two crucial pieces of information you need for BASIC are: 1) the graphics mode in which BASIC should assume you are working (it will be different for different rows on the screen), and 2) the location in memory that BASIC should consider the beginning of screen RAM for that particular portion of the screen.

Diagram 3



These locations can be arbitrarilly chosen so long as they point to free areas in memory.

For example, if you have a screen with Graphics 2, Graphics 3, and Graphics 0 on it, you need three separate pieces of memory where the information for each portion of the screen can be stored (see Diagram 3). Now, if you want to put something into your Graphics 3 area on the screen, you can put numbers directly into screen RAM (starting at location 40800 in this example) as we did earlier. Or, you can use BASIC PLOT and DRAWTO statements (much easier) by first telling BASIC that you are working in Graphics 3 and that screen RAM starts at location 40800. Then, for example, if you say PLOT 0,0 you will get a colored point at the position marked by the arrow (-->) to the left of the diagram. To see how this all works, follow the instructions on Display List Worksheet #3.

Display List Worksheet #3

The first thing we are going to do is create a display list with Graphics 2 at the top of the screen, then Graphics 3, and finally Graphics 0. For now, there will be little explanation of how we create this new display list. (You will learn more about how to do this in the next section.) In this section, you will be concentrating on how to get BASIC to print to different modes on different portions of the screen.

We will start with a Graphics 3 (split screen) display list and modify it by putting Graphics 2 at the top. The standard Graphics 3 display list and the modified version that we want to create are shown below:

GR. 3	112	112	112	72	112	158	8	8	8	8	8	8	8 ...
Modified	112	112	112	71	112	158	7	7	72	172	158	8	8 ...

The following program will create the modified display list. Type it in and run it.

```
10 DIM A(10)
20 GR. 3
30 DLIST=256*PEEK(561)+PEEK(560)
40 FOR I=0 TO 5
50 A(I)=PEEK(DLIST+I)
60 NEXT I
70 A(3)=71:A(6)=7:A(7)=7:A(8)=72:A(9)=172:A(10)=158
80 FOR I=0 TO 10:POKE DLIST+I,A(I):NEXT I
90 END
```

Now let's be sure we really have both Graphics 2 and Graphics 3 on our screen. Try the following:

```
LOC=158*256+112     (This is screen RAM for Graphics 2)
POKE LOC,33         (33 is the code for "A")
POKE LOC+20,34
POKE LOC+40,35
```

Hopefully, "A", "B", and "C" showed up in Graphics 2.

Try POKEing other numbers into screen RAM. Then try:

```
LOC=LOC+60          (Now LOC is the address of
POKE LOC,255         Graphics 3 RAM)
POKE LOC,75
```

Hopefully, some blocks of light showed up in Graphics 3.

Now let's see about the bottom part of the screen.

```
LOC=159*256+96
```

(Press SHIFT-CLEAR so that the cursor is at the top of the window to prevent the window from scrolling.)

```
POKE LOC,33
```

Now let's try to get BASIC to do some of the work.

```
POKE 87,2
```

(In location 87 we POKE the graphics mode we want BASIC to think we're using).

```
POKE 88,112:POKE 89,158
```

(Locations 88 and 89 are used to store the two numbers which point to screen RAM. Now we can use regular BASIC statements).

```
PRINT #6;"your name"
POSITION 10,0
PRINT #6;"WOW"
```

(Switch to Graphics 3 area).

```
POKE 87,3
POKE 88,172:POKE 89,158
```

(The numbers 172 and 158 are the two numbers which define the starting location of the Graphics 3 RAM. The actual location in memory is 158*x56+172).

```
COLOR 1
PLOT 0,0:DRAWTO 10,0
COLOR 2:DRAWTO 10,10
COLOR 3:DRAWTO 0,10
```

Just for the fun of it, try the following:

```
POKE 87,2
```

(BASIC now thinks we're in mode 2, but we're still pointing to Graphics mode 3 RAM).

```
POSITION 0,0
PRINT #6;"your name"
```

Experiment with this for a while.

Customized Display Lists

In this section you will learn about scan lines and mode lines so that you can design your own display lists. You will also learn about organizing and addressing memory so that you can use your display lists.

Depending on which graphics mode you are using, you can have a different number of lines of information on the screen. For example, in Graphics 0 you can have 24 lines (rows) of text. We call each of these lines a <u>mode line</u>. In GRAPHICS 2, you can only fit 12 lines of text on the screen. Graphics 2 has 12 mode lines per full screen.

Each mode line has a certain width (perhaps easier to think of as a height) to it. Graphics 0 mode lines are thinner than Graphics 2 mode lines. In fact, they are twice as thin, which is why you can get twice as many lines on a full screen in Graphics 0. We count the thickness of mode lines in terms of a fixed width called a <u>scan line</u>. A Graphics 0 mode line has 8 scan lines while a Graphics 2 mode line has 16 scan lines (twice as thick).

The entire screen can hold up to 192 scan lines, but no more. Since each Graphics 0 line has 8 scan lines, you can have 192/8 = 24 mode lines on the screen in Graphics 0. In contrast, each Graphics 7 mode line uses up only 2 scan lines. Thus you get 192/2 = 96 Graphics mode 7 lines on one screen. (See chart at end of this section: "Antic and the Display List").

When you construct your own display list, you must keep count of how many scan lines you are using and be sure not to go over 192.

In the previous section you learned that there needs to be screen RAM devoted to each portion of the screen. Depending on which graphics mode you use, each mode line uses up a different amount of memory. Memory is counted by counting things called bytes. The "Antic and the Display List" chart shows how many bytes are used up by each mode line. For example, it takes 40 bytes for each Graphics 0 mode line. If you want a full screen of Graphics 0, it takes 24 mode lines x 40 bytes for each mode line = 960 bytes of

memory.

To help you understand all of this better, take the time to do Display List Worksheet #4 now.

Display List Worksheet #4

For each of the following screen displays, fill in the numbers where a blank exists. All graphics modes are printed as BASIC, not Antic, modes. You'll need to use the "Antic and the Display List" chart. You'll also need to use the fact that a full screen has 192 scan lines.

1.

GR. 0	8 mode lines (64 scan lines)	320 bytes RAM needed
GR. 3	___ mode lines (128 scan lines)	___ bytes RAM needed

2.

GR. 2	___ mode lines	100 bytes RAM needed
GR. 3	14 mode lines	140 bytes RAM needed

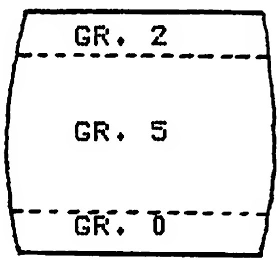
3.

GR. 7	___ mode lines	___ bytes RAM needed
GR. 1	4 mode lines	___ bytes RAM needed

4.

GR. 2	2 mode lines	___ bytes RAM needed
GR. 5	___ mode lines	___ bytes RAM needed
GR. 0	2 mode lines	___ bytes RAM needed

It's finally time to make your own display list. Let's use problem 4 on worksheet #4 as an example. If you correctly solved the problem, you should have the following:



2 mode lines 40 bytes RAM needed
(2 x 20 bytes per mode line = 40)

36 mode lines 720 bytes RAM needed
(36 x 20 bytes per mode line = 720)

2 mode lines 80 bytes RAM needed
(2 x 40 bytes per mode line = 80)

We always begin our display list with 112 112 112. Each of these numbers tells Antic to skip 8 blank lines for a total of 24 scan lines. We need to skip these because they are above the visible screen -- this is called overscan, but all you need to worry about is that you always start your display list this way.

You must decide where in memory to begin your screen RAM. You need a total of 40 + 720 + 80 = 840 bytes. Next, calculate how many <u>pages</u> of memory you need (1 page = 256 bytes). Thus 840/256 = 3 + extra. So, 4 pages of memory will hold all of screen RAM. It is customary to put screen RAM near the end of memory. Calculate backwards from 159 pages (on a 48K machine) -- 159 - 4 pages = 155 pages. This calculation shows that screen RAM should begin on the first byte (called byte 0) on page 155. This pair of numbers (0, 155) is called a <u>memory address</u>. We say the <u>low byte</u> is 0 and the <u>high byte</u> is 155.

The first graphics mode is BASIC mode 2, but you must always use the Antic mode number which is 7 in this example. The display list must tell the computer that there is an Antic mode 7 line and it must also signal that the next two numbers are the address of screen RAM (low byte, then high byte). You may recall that this is done by adding 64 to the Antic code. So now we have ...

```
112
112
112
 71      71 = 64 + 7.  The first Antic mode 7 line
  0      followed by the two numbers addressing
155      screen RAM.
  7      7 is the second Antic mode 7 line.
```

So above the two mode lines of BASIC mode 2 are defined in the display list. To continue, you must change to BASIC mode 5 (Antic mode 10). First, calculate where screen RAM will start. Let's put it after screen RAM for the Graphics 2 lines. Since Graphics 2 RAM starts at byte 0 on page 155 and it uses up 40 bytes, you can start Graphics 5 RAM at byte 40 on page 155. So we have ...

```
112
112
112
 71
  0
155
  7
 74      74 = 64 + 10.  The first Antic mode 10
 40      (BASIC mode 5) line followed by
155      the screen RAM address.
 10
 10      Here you need 35 more 10's to specify the
 10      remainder of the 36 Antic mode 10 lines.
  .
  .
```

Now where should your Graphics 0 (Antic mode 2) screen RAM begin? For Graphics 5, you started at byte 40 on page 155 and used up 720 more bytes.

40 + 720 = 760 bytes.
760/256 = 2 pages with 248 bytes extra.
So the starting address is byte 248 on page 155+2 = 157.

Now we have ...

```
112
112
112
 71
  0
155
  7
```

```
 74
 40
155
 10      You need 35 tens.
 10
  .
  .
 10
 66      66 = 64 + 2.  The first Antic mode 2 line.
248
157
  2      The second (last) mode 2 line.
 65      Below is an explanation of
  0      these last three numbers.
159
```

The third from last number in the display list is always 65 which is always followed by another address. This last address (0, 159) is where you decide to actually place the display list itself. You can fit it on page 159 as long as there are no more than 255 numbers in the display list.

The remarks on the following program provide you with the necessary explanations to put your display list into memory and use it. This program is stored in a file called "EXAMPLE.DLS". The remarks are for your own understanding, so study them carefully. Then try running the program.

```
100 REM *      Start with a GRAPHICS 0 statement or BASIC gets
110 REM *      confused when you try to print to your custom
120 REM *      display list.
130 REM *
140 GRAPHICS 0
150 REM *
160 REM *      This next statement turns off the screen.  When
170 REM *      changing display lists, it's a good idea to turn
180 REM *      off the screen while you make the changes.
190 REM *
200 POKE 559,0
210 REM *
220 REM *      This is the location where we are going to put
230 REM *      our display list.
240 REM *
250 DLIST=256*159
260 REM *
```

```
270 REM *      The next lines POKE our display list into memory
280 REM *      on page 159.
290 REM *
300 FOR I=0 TO 51
310 READ CODE:POKE DLIST+I,CODE
320 NEXT I
330 DATA 112,112,112,71,0,155,7,74,40,155
340 DATA 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
350 DATA 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
360 DATA 66,248,157,2,65,0,159
370 REM *
380 REM *      Next we need to clear screen RAM with zeros.
390 REM *
400 FOR I=155*256 TO 159*256-1
410 POKE I,0
420 NEXT I
430 REM *
440 REM *      These POKEs tell the computer where our display
450 REM *      list is located in memory.  We POKE the two
460 REM *      numbers of our address into locations 560 and 561.
470 REM *
480 POKE 560,0:POKE 561,159
490 REM *
500 REM *      Turn the screen back on.
510 REM *
520 POKE 559,34
530 REM *
540 REM *      We'll use GRAPHICS 2 first.
550 REM *
560 POKE 87,2
570 REM *
580 REM *      And we POKE the beginning address of GR. 2 RAM.
590 REM *
600 POKE 88,0:POKE 89,155
610 PRINT #6;"   WOW!"
620 PRINT #6;"   this is neat"
630 REM *
640 REM *      Let's use GRAPHICS 0 now and tell BASIC where
650 REM *      screen RAM begins for this mode.
660 REM *
670 POKE 87,0
680 POKE 88,248:POKE 89,157
690 POSITION 2,0:PRINT "My very own"
700 PRINT "Custom Display Screen."
710 REM *
720 REM *      And finally, GRAPHICS 5.
730 REM *
740 POKE 87,5
750 POKE 88,40:POKE 89,155
760 COLOR 1:PLOT 0,0:DRAWTO 79,35
770 COLOR 2:PLOT 79,0:DRAWTO 0,35
780 GOTO 780
```

Summary and Challenges

Important Display List Memory Locations

560, 561	Low byte, high byte address of where display list is located in memory. BASIC supplies these numbers when it creates a display list. If you create one, you must indicate its location in these bytes.
87	POKE with a 0 through 8. Tells BASIC what graphics mode you're working in.
88, 89	Low byte, high byte address of screen RAM that BASIC uses when printing and plotting.

Caution! There are some special cases (especially when you use GRAPHICS 8) in which screen RAM uses so much memory that a special code must be placed in the display list for it to work. This happens when screen RAM crosses a special boundary in memory called a 4K boundary. These boundaries occur at page 144, page 128, page 112, and at subsequent 16 page intervals. You need not worry about this if these boundaries never show up in the middle of your screen RAM. If this is unavoidable, refer to _Your Atari Computer_ by Poole, McNiff and Cook on pages 300 - 301.

Finally, you should be aware that there is no reason to do all of the hard work that is required every time you want to implement your own display list. Software exists that does all of the calculating and memory management for you, leaving you free to devote your efforts to creating interesting screen images. You may wish to explore "SCREEN MAKER" by Wayne Harvey to see how such a tool can save you time and trouble.

As a final challenge, try to design an interesting title screen for some type of program (a game or other project). Start by laying out your screen image on paper and then create the necessary display list. Finally, write a BASIC program to finish the task.

Antic mode	Basic mode	Mode lines/ full screen	Scan lines/ mode line	Bytes/ mode line
2	0	24	8	40
3	none	19	10	40
4	none	24	8	40
5	none	12	16	40
6	1	24	8	20
7	2	12	16	20
8	3	24	8	10
9	4	48	4	10
10	5	48	4	20
11	6	96	2	20
12	none	192	1	20
13	7	96	2	40
14	none	192	1	40
15	8	192	1	40

DISPLAY LIST INTERRUPTS

In order to implement a Display List Interrupt, there are four steps that must be completed:

1. Decide where on the screen the interrupt should occur, and set the Interrupt enable bit of that line of the Display List.

2. Write the Assembly Language code that is to be executed during the interrupt, and place it in memory.

3. Tell the computer where the routine is located by pointing the DLI vector to the code's location in memory.

4. Enable Display List Interrupts by setting the proper bit in the Interrupt Mask location.

ALTERING THE DISPLAY LIST

To cause a DLI on a specific mode line of the Display List, add 128 to the mode line's value. As an example, to cause a DLI on the 5th line of a Graphics 0 screen, find the 5th mode line in the Display List:

```
112  blank 8 lines
112  blank 8 lines
112  blank 8 lines
66   use graphics 0, start memory at
64   low byte of screen RAM address
156  high byte of screen RAM address
2    2nd GR 0 line
2    3rd
2    4th
2    the 10th number in the Display List is mode line 5
```

Add 128 (set the highest bit) to the 2 from this line to get 130. Poke this 130 back into the Display List, and when the DLI is enabled, it will occur on this line of the screen.

WRITING THE INTERRUPT CODE

Display List Interrupt Assembly code must be fairly short because of time limitations. Because the subroutine may be called at any time, any registers that are affected by the routine must first be saved, and then restored before leaving the routine. This is commonly done by using the stack and the instructions PHA and PLA. If the X or Y register is needed, use the TXA or TYA commands in conjunction with the PHA, and reverse the process to restore X or Y using PLA and TAX or TAY. It is best to use only the Accumulator if possible. The routine should end with the Return From Interrupt command - RTI. Do not use the RTS command. If you do not want the changes you make to take effect until the end of the scan line being drawn on the TV, you must include the command STA WSYNC, where WSYNC = 54282. Any value in the Accumulator will do the job. The following sample DLI turns the bottom of a Graphics 0 screen to pink. (the X register needn't be used, but is used for demonstration purposes):

```
PHA         save accumulator
TXA
PHA         save X register
LDA #80     will make the characters dark ( 0 luminance)
LDX #88     color pink
STA 54282   ( WSYNC ) delays color changes until end of line
STA 53271   store dark chars in hardware register
STX 53272   store pink in background hardware register
PLA
TAX         restore X register
PLA         restore Accumulator
RTI         Return From Interrupt
```

It is important to understand the concept of Hardware and Shadow registers. At the end of the drawing of every TV screen, the computer stops what it is doing and copies several shadow register values back into corresponding hardware registers. An example of this is the color registers. A value POKED into 708 will be moved to 53270 at the end of each screen. This means that if the program changes 708, the effects will not be seen until the end of that screen. Also, the effects will be permanent, because the computer will keep moving 708's value into 53270. Therefore, if you want a change to take place immediately, and to end at the end of the screen, you must make your changes to the hardware registers. (in this case, to 53270)

THE DLI VECTOR

Once your code is written and stored safely in RAM, you must tell the computer where it is located. There are two locations which form a vector which identifies this RAM location. Locations 512 and 513 (hex $200 and $201) are the low and high bytes, respectively, which can be combined to represent any Atari RAM location. For example, if your code was located at the beginning of page six in memory (starting at hex location $0600), the two bytes would be as follows:

```
512 = 0    the low byte
513 = 6    the high byte
```

If your code was stored at location 35000 in RAM, you would calculate the values for 512 and 513 by dividing 35000 by 256:

```
           136   is the quotient (high byte)
       -------
 256 ) 35000
        256
        -----
         9400
         768
         ----
         1720
         1536
         ----
          184   is the remainder (low byte)

512 = 184
513 = 136
```

ENABLING YOUR DLI

After completing the three previous steps, you are ready to enable your Display List Interrupt. To do so, all that is required is one poke into location 54286. The following will do the the job:

```
From Basic:     POKE 54286 , 192

From Assembly:  LDA #$C0
                STA $D40E
```

To disable the interrupts, poke the same location with 64 ($40).

VERTICAL BLANK INTERRUPTS

A Vertical Blank Interrupt routine is executed after each drawing of the television screen (60 times a second). During a VBI the programmer can execute as many as 5,000 machine language commands. There are several advantages to being able to execute this much code in this fashion. Because the TV screen is not being drawn during a VBI, any changes to screen RAM or to Player/Missile Graphics will not be seen until the next drawing of the screen. This can eliminate some unsightly flickering, and allow for smooth full-screen scrolling. In some ways VBIs are more complicated than Display List Interrupts, and yet they are somewhat easier to use. It is important to understand what goes on at the end of each TV screen's drawing before attempting to write and use a VBI.

At the end of the screen the computer saves the A, X, and Y registers on the stack (this means you will not have to do this yourself) and goes to the Immediate Vertical Blank Routine. It finds this routine by using the vector at locations $222 and $223. Normally these locations point to an Operating System routine located at $E45F. This is the routine that copies the colors from the shadow registers into the hardware registers, reads the joystick, increments the system clock, etc. This routine ends with an inderect jump command through another vector located at $224 and $225. This points to the Defered Vertical Blank Routine, and the command in Assembly language is JMP ($224). Usually this vector points to a small routine at $E462 that restores A, X, and Y and does an RTI.

For most purposes, the Deferred VBI is the time to run our code. The steps to set up this VBI are:

1. Write the code and store it memory.

2. Change the vector that points to the Deferred VBI.

If you desire to have your code executed before the Operating System routine, then you must read the section of De Re ATARI which deals with the Vertical Blank Interrupt. See pages 8-16 to 8-19.

THE CODE

Write Assembly code for the task you want to accomplish, and place it in RAM. The only restrictions are the length of time your code takes to run (20,000 machine cycles - see Levanthal's 6502 book for cycle counts), and in the way the code must end. Your last statement must be a JMP $E462. This will restore the registers and do the RTI. Your code will not execute until the vector at $224 and $225 is pointed to its location in RAM.

THE VECTOR

The Deferred VBI vector must be changed very carefully. If only one of the two bytes is reset to the desired location, then the address contained in the vector is innaccurate, and should the interrupt occur at this point, the computer will go to the wrong location in memory to find the code. Chances are good you will crash the machine. The vector must be changed quickly (not from BASIC) and at a safe time in the drawing of the TV screen. The Operating System has just the routine to do the job, but it must be called from Assembly language. To use the routine, load the low byte of your code's location into the Y register. Next, load the high byte of the location into the X register. Now load the accumulator with a 6 for a immediate VBI, or a 7 for a deferred VBI. Finally, do a Jump-Subroutine to SETVBV ($E45C). Following is an example which sets the vector to point to a routine located at the beginning of page six ($600) which is to be executed in the deferred VBI:

```
LDY #0     low byte of address
LDX #6     high byte of address
LDA #7     means deferred VBI
JSR $E45C change the vector
```

You will probably need to modify this routine to be used from the USR function in BASIC.

THE USR COMMAND

BASIC allows you to write assembly language programs and run the resulting machine language routines using the USR command. The command follows the format:

A = USR (address , param1 , param2 , ...)

The address in memory where the machine language instructions are located is the first number after the left parenthesis. It is optional to then include 1, 2, or more other values before the right parenthesis. These optional numbers can be used to pass information into the machine language routine. The variable A may contain a value returned by the machine language routine, if the programmer so desires. The address and optional parameters may be constants or variables, and are converted into integers by BASIC. They may not be negative numbers.

When BASIC turns control over to the machine language routine, it saves the A, X, and Y registers, and places on the stack the address in the BASIC code to which it hopes to return. To allow it to do so your routine must end with the command Return From Subroutine (RTS). Also each of the parameters you included in BASIC is converted into a two byte number which is placed on the stack, low byte followed by high byte. After each of the parameters has been placed on the stack, BASIC puts one more number on the stack which indicates how many parameters have been passed. This means that even if no parameters have been passed, you must pull at least one number off of the stack before attempting the RTS. Failure to do so will cause BASIC to return to a bad address when your routine is done, and you may very well crash the computer. In the same way, if you pull too many numbers off of the stack, the return address will also be invalid. Be careful, and if your routine crashes the system, look at your stack pulling as a first possible cause.

For a sample program, assume we wanted to use the speed of machine language to search the screen to count the number of occurances of any character we choose. Because we want our routine to be versatile, we want to be able to specify where in RAM our screen begins (in case of different RAM sizes, or multiple screens). This would mean we will need two parameters: The RAM location at which to begin, and the character which we are counting (we will have to pass the internal code number which represents our character). The routine itself will compare every byte of screen RAM and return the count in our variable to the left of the "=" in our USR call. We will store our routine in page six of memory (beginning at decimal location 1536) and we will have to use several page zero memory locations for our pointers and calculations. In BASIC we may use locations 203 through 209. Locations 212 and 213 are used to return the value from the USR command. Our program is on the next page. (The only important thing to understand is how to pull your parameters off of the stack and save them, and how to return a value to BASIC and leave the subroutine. Don't worry about the logic of the counting routine.)

(As a final note, always save any program using USR before running it. There are so many things that can go wrong, you can afford the minute it takes to save your work.)

To call our routine, use the command:

```
COUNT = USR ( 1536 , 40000 , 10 )
```

This call searches the screen beginning at 40000 in RAM for the character "*". See page 55 of the BASIC Reference Manual for character numbers. 40000 is usually the beginning of screen RAM in a 48K system after pressing the System Reset Key.

```
; SUBROUTINE TO COUNT THE OCCURANCES OF A GIVEN CHARACTER
; IN GRAPHICS ZERO SCREEN RAM BEGINNING AT A SPECIFIED LOCATION
; TO BE CALLED FROM THE USR COMMAND IN ATARI BASIC
; USE THE FORMAT:
;
;         COUNT = USR ( 1536 , SCR-RAM-ADR , CHAR-NUM )
;
COUNT   = 212                  Return value from USR command
POINTER = 203                  Used for Indirect, Indexed Addressing
CHAR    = 205                  Saving the character we're looking for
        *=$0600
;
INIT    PLA                    Number of parameters - should be 2
        PLA                    High byte of screen RAM address
        STA POINTER+1          High byte stored second
        PLA                    Low byte of screen RAM address
        STA POINTER            Low byte stored first
        PLA                    Char high byte should be zero - ignore
        PLA                    Char low byte
        STA CHAR
        LDA #0
        STA COUNT              Both bytes of return value
        STA COUNT+1            set to zero
        LDX #4                 We are going to search counting
;                              4 times 240 = 960 bytes per screen
LOOPBIG LDY #240
;
LOOP    LDA (POINTER),Y        Get each character
        CMP CHAR               Is it the kind we are counting?
        BNE SKIP               If not then move on
;
        LDA COUNT              If it is, then add one to the low
        CLC                    byte of our COUNT
        ADC #1                 Use the Carry bit to alter the
        STA COUNT              high byte of our count, if neccessary
        LDA COUNT+1
        ADC #0
        STA COUNT+1
;
SKIP    INY                    Next character location
        CPY #240               Up to 240, then we have to alter
        BNE LOOP               our pointer to Screen RAM
;
        LDA POINTER            Add 240 to the low byte, and fix
        CLC                    the high byte using the Carry Bit
        ADC #240
        STA POINTER
        LDA POINTER+1
        ADC #0
        STA POINTER+1
        DEX                    Count down until we have done it
        BNE LOOPBIG            Four times, and then Return to BASIC
        RTS
                               .END
```

ATARI 1020 PRINTER
****

First of all, open the printer:

```
OPEN #1,8,0,"P:"
```

To get into GRAPHICS mode:

```
PRINT #1;"(ESC) (ESC) CTRL G"
```

To return to TEXT mode:

```
PRINT #1;"A"
```

THE FOLLOWING COMMANDS ARE GIVEN FROM THE TEXT MODE

To get 20 characters per line:

```
PRINT #1;"(ESC) (ESC) CTRL P"
```

To get 80 characters per line:

```
PRINT #1;"(ESC) (ESC) CTRL S"
```

To get back to 40 characters per line:

```
PRINT #1;"(ESC) (ESC) CTRL N"
```

To change the SCALE of the letters:

```
PRINT #1,"S(a value from 0 to 63)"
(0 is small, 63 is huge)
```

To access the INTERNATIONAL character set:

```
PRINT #1;"(ESC) (ESC) CTRL W"
```

To return to the STANDARD character set:

```
PRINT #1;"(ESC) (ESC) CRTL X"
```

THE FOLLOWING COMMANDS ARE GIVEN FROM THE GRAPHICS MODE

To return the pen to the HOME position:

```
PRINT #1;"H"
```

To specify the pen COLOR:

```
PRINT #1;"C(a value from 0 to 3)
(0=black, 1=blue, 2=green, 3=red)
```

To specify the LINE TYPE:

```
PRINT #1;"L(a number from 0 to 15)
(0 is a solid line, 15 is dots, and 7 is dashes)
```

To INITIALIZE the printer, or set the current position as 0,0:

```
PRINT #1;"I"
```

To DRAW to a specific point:

```
PRINT #1;"Dx,y"
(x is a value from 0 to 480, and y is from -999 to 999)
```

To DRAW RELATIVE to your current position:

```
PRINT #1;"Jx,y"
(x and y are the same as for DRAW)
```

To MOVE to a specific point:

```
PRINT #1;"Mx,y"
```

To MOVE RELATIVE to your current position:

```
PRINT #1;"Rx,y"
```

To draw a Y AXIS:

```
PRINT #1;"X0, distance between tics, number of tics"
```

To draw an X AXIS:

```
PRINT #1;"X1, distance between tics, number of tics"
```

To ROTATE the text by 90 degrees:

```
PRINT #1;"Q(a value from 0 to 3)
(0 is normal, one is down, etc. clockwise)
```

To repeat the same command in one PRINT statement:

Do not repeat the code letter,
and separate the values with a ;
ex. PRINT #1;"D10,10;30,30;100,-100"

To give successive commands in one PRINT statement:

Separate the commands with a *
ex. PRINT #1;"C2*M10,20*D100,100"

Wherever the phrase "a value from 0 to #" appears, either an integer value or a variable may be used in the PRINT command.